ECE 438: Computer Design
Electrical and Computer Engineering
University of New Mexico

Instructor: Andrew Targhetta

Lab 3

A Single-cycle MIPS Processor

Due: April 2nd, 2020

Introduction

The objective of the third ECE438 laboratory exercise is to describe a simple, *single-cycle* MIPS processor in VHDL or Verilog if you prefer. In future labs, we will pipeline our MIPS processor; however, for now, the single-cycle design will allow us to work out any issues with the components of our datapath.

Background

The MIPS ISA is part of the Reduce Instruction Set Computer (RISC) design philosophy that simplifies instruction decoding with a small number of fixed-width instruction formats coupled with the load-store architecture concept. As a result, the MIPS ISA is a great teaching tool for understanding the basic concepts of computer design. In this lab, you will implement a subset of the MIPS ISA that is enough to run useful programs without unnecessary design complexity. In future labs, you will expand your microarchitecture to support more of the MIPS ISA. For now, we will support the following instructions:

- Arithemtic — ADDU, ADDIU, SUBU
- Logical — AND, ANDI, NOR, OR, ORI, XOR, XORI

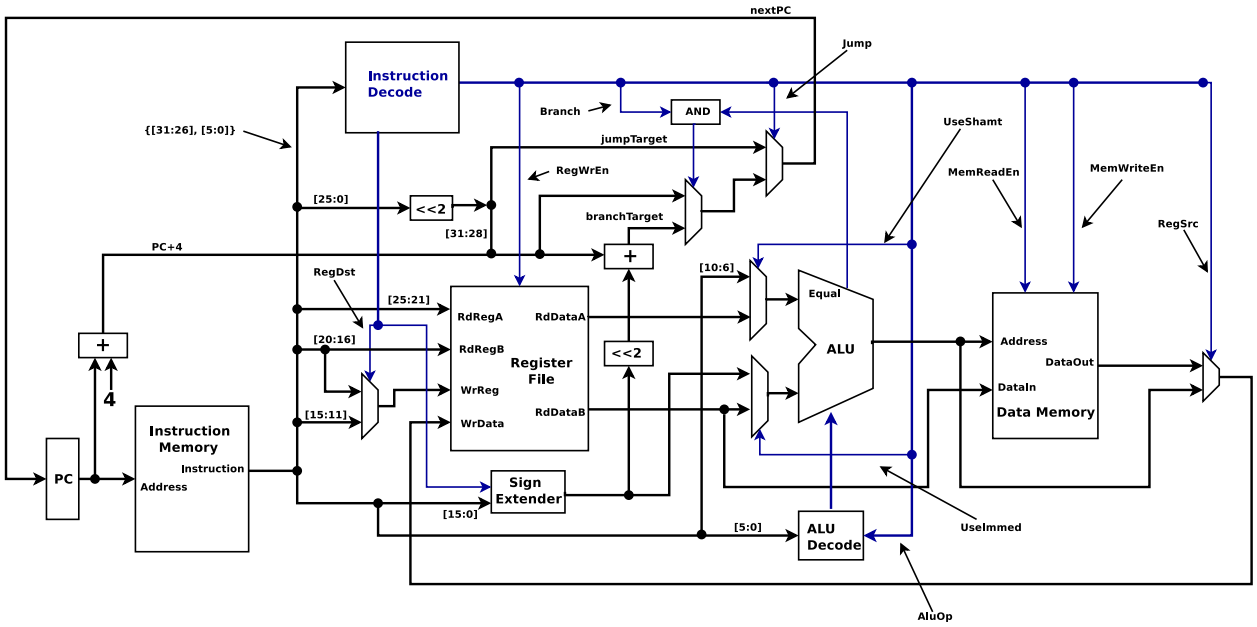

Figure 1: Single-cycle MIPS Block Diagram

- Shift — SLL, SRL, SRA
- Comparison — SLTU, SLTIU
- Loads and stores — LW and SW
- Branch Equal and Jump — BEQ and J
- Load Upper Immediate — LUI

Figure 1 shows a block diagram of the single-cycle MIPS processor you will design in lab. The register file, decoder, ALU, and ALU control should have been described in individual VHDL components during previous lab exercises. The remaining datapath items depicted in Figure 1, with the exception of the instruction and data memory, will be described in a single VHDL file (`SingleCycleProc.vhd`) in this lab. The ports to the instruction and data memory will be brought out of the `SingleCycleProc` component and instantiated in a top-level file provided for you. For the single-cycle design, you will not support a *branch delay slot*. We will add this support in when we pipeline our processor.

Procedure

1. Describe the datapath depicted in Figure 1 using the skeleton code provide below as a starting point. `PC` in your design should be set to 0 when `Rst_L` goes low and a rising edge of the `Clk` occurs. In other words, 0 is the reset address of your processor, and `Rst_L` is an active-low, synchronous reset.

```
1  ----------------------------------------------------------------------------
   -- This VHDL source file describes the basic single-cycle processor
3  -- discussed in David Patterson and John Hennessy's Computer Organization and
   -- Design textbook.
5  ----------------------------------------------------------------------------

7  library ieee;
   use ieee.std_logic_1164.all;
9  use ieee.std_logic_unsigned.all; --needed to describe
   use ieee.std_logic_arith.all;    --arithmetic
11 ----------------------------------------------------------------------------

13 entity SingleCycleProc is

15 port( Clk, Rst_L: in std_logic;
         PC: out std_logic_vector(31 downto 0);
17       Instruction: in std_logic_vector(31 downto 0);
         DataMemAddr: out std_logic_vector(31 downto 0);
19       DataMemRdEn, DataMemWrEn: out std_logic;
         DataMemRdData: in std_logic_vector(31 downto 0);
21       DataMemWrData: out std_logic_vector(31 downto 0)
   );
23 end SingleCycleProc;

25 ----------------------------------------------------------------------------

27 architecture SingleCycleProc of SingleCycleProc is

29 -- Component declarations go here...

31     component RegisterFile is
       port( RdRegA: in std_logic_vector(4 downto 0);
33           --insert code here for read port B
             WrReg: in std_logic_vector(4 downto 0);
35           Clk: in std_logic;
             --something else is missing here...
37           WrData: in std_logic_vector(31 downto 0);
             RdDataA: out std_logic_vector(31 downto 0);
39           --insert code here for read port B
       );
41     end component;

43 -- Declare signals and variables here...
```

```
45 -- Typically, the code is easier to read if signals are
   -- declared in the order they appear below...
47 signal Rs, Rt, Rd: std_logic_vector(4 downto 0);
   signal AluInA: std_logic_vector(31 downto 0);
49 signal AluInB: std_logic_vector(31 downto 0);
   signal AluResult: std_logic_vector(31 downto 0);
51 signal Equals: std_logic;
   signal RegDst: std_logic;
53 signal WrReg: std_logic_vector(4 downto 0);
   signal RegWrData: std_logic_vector(31 downto 0);
55 --fill in the rest here

57 ---------------------------------------------------------------------------
   begin
59
     --describe PC logic here
61
       --break up instruction into parts for readability
63     Rs <= Instruction(25 downto 21);
       Rt <= Instruction(20 downto 16);
65
       --instantiate the register file
67     --instance_name: component_name
       -- use nominal port map as opposed to positional!
69     Registers: RegisterFile
           port map(RdRegA => Rs,
71                  RdRegB => Rt,
                    WrReg => WrReg,
73                  Clk => Clk,
                    WrData => RegWrData,
75                  --okay you get the point...
           );
77


79
       --zero extend the shift amount field
81     Shamt(4 downto 0) <=  Instruction(10 downto 6);
       Shamt(31 downto 5) <= (others => '0');
83
       -- ALU source mux for OpA
85     --The WITH/SELECT/WHEN construct is great for muxes
       with UseShamt select
87         -- the zero extended shamt field
           AluInA <= Shamt when '1',
89         -- the register file output
                  RdRegA when others;--avoids latches!
91
       -- ALU should be instantiated somewhere here...
```

```
93
      --Describe register destination mux using when/else
95    WrReg <= Rd when (RegDst='1') else --finish this line

97 end SingleCycleProc;
   ---------------------------------------------------------------------------
```

2. Using the provided testbench, verify the correct operation of your processor.

1 Deliverables

1. Email all VHDL source files with comments to `adtargh@unm.edu` .

   Note: Code submitted without adequate comments will not be graded!

2. Answer the following questions:

   (a) Which components in Figure 1 does the testbench file, `SingleCycleProc_tb.vhd`, provide?

   (b) The testbench uses MIPS assembly to test your processor. Examine the assembly in the testbench and explain in your own words how it exercises your processor.

   (c) Is the testbench code *synthesizable*? Why or why not?

   (d) The existing testbench code is written assuming branch delay slots are not supported. Modify the testbench to support branch delay slots.